Universidad Simón Bolívar
Departamento de Computación y
Tecnología de la Información
CI-3825: Sistemas de Operación I

Prof. Fernando Torre Mora

Laboratorio de Semana 3

Scout Sort

1 Introducción

Uno de los mayores problemas al realizar un sort de un arreglo de números, es la verificación de que los números se encuentren ordenados. Para solucionar este problema se propone Scout Sort, un algoritmo de ordenamiento que crea procesos "scout" que revisarán puntos designados del arreglo e indicará al programa padre si necesitan ser intercambiados.

2 Funcionamiento

El algoritmo comienza dividiendo el array en dos mitades iguales y asignándole a cada hijo una mitad, indicada por los índices que debe revisar. Si el hijo recibe 1 elemento, debe retornar EXIT_SUCCESS; en caso contrario, repite el procedimiento. Si ambos hijos retornan EXIT_SUCCESS, compara los números en la frontera y, si están en orden, supone que todo el arreglo asignado está en orden y retorna EXIT_SUCCESS.

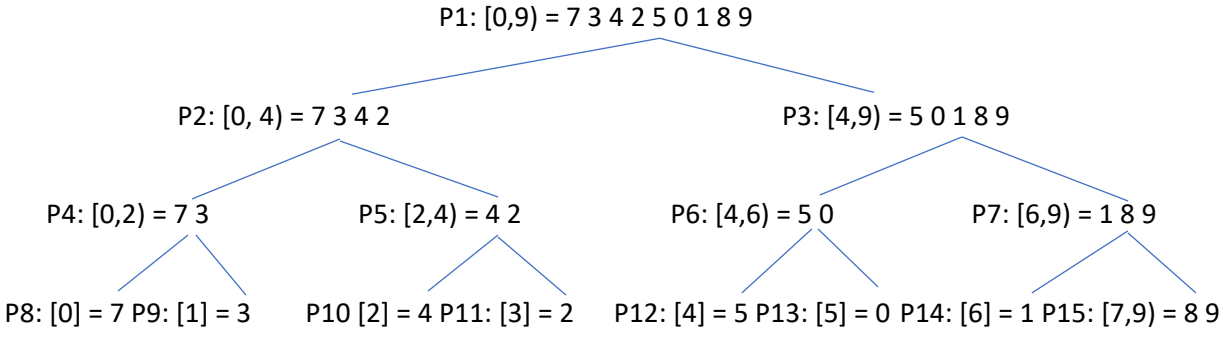
Si los números no están en orden, retorna el número de elementos (ya que el proceso hijo solo tiene una copia del array siendo ordenado, no puede realizar el cambio por sí mismo; este retorno le indica al padre que debe tomar la acción correspondiente)

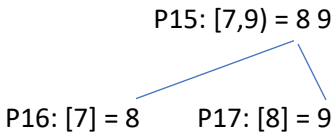
Si algún hijo no retorna EXIT_SUCCESS, debe pasarse este valor al padre como EXIT STATUS.

Al llegar al padre, si ambos hijos directos retornan EXIT_SUCCESS, y los valores de la frontera están en orden, sabrá que el arreglo completo está ordenado. Si no, esto le indica los posibles elementos problema (un hijo que retorne 4 indica que el 3er y 4to valor, o el 7moy 8vo valor, etc, no se encuentran en orden). El programa padre entonces revisa sólo estas parejas de valores, intercambiándolos donde sea necesario, y luego repite el procedimiento.

3 Ejemplo

Suponga que se desea ordenar la lista 7 3 4 2 5 0 1 8 9; con 9 elementos, la lista se divide en los primeros 4 y los últimos 5 elementos. Estos se subdividen hasta llegar a 1 elemento, en cuyo momento comienza a retornar.

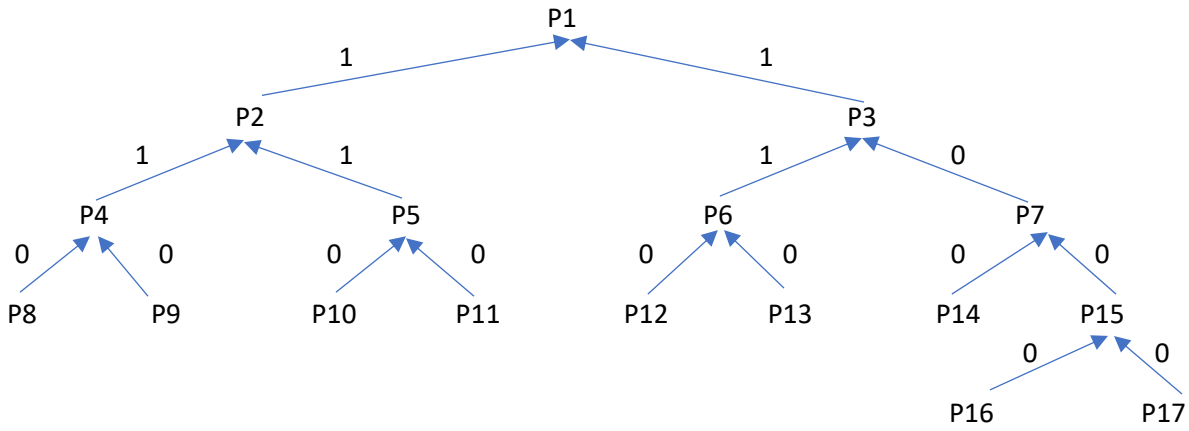




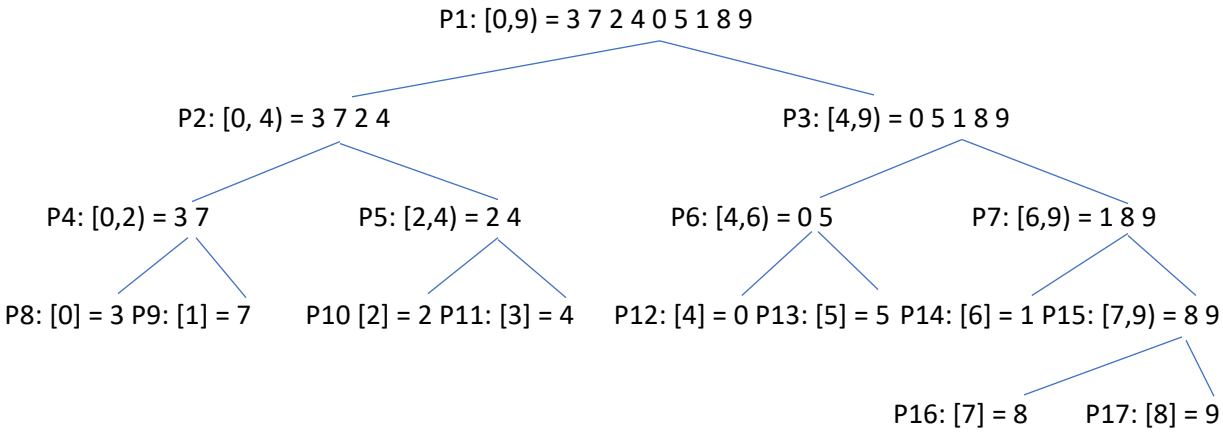
Los procesos 8 – 14, 16 y 17 retornan de inmediato, dejando a los procesos 4, 5, 6 y 15 en necesidad de comparar, respectivamente, 7 vs 3, 4 vs 2, 5 vs 0, y 8 vs 9. Los primeros tres están desordenados, por lo que retornan 1, haciendo que P2 retorne 1, indicándole a P1 que debe revisar e intercambiar las posiciones 0 con 1, y 2 con 3.

Mientras tanto, P15 retorna EXIT_SUCCESS, ya que sus números están en orden, e indicándole a P7 que debe revisar la frontera: las posiciones 6 y 7. Estas están, en efecto, en orden, por lo que P7 retorna EXIT_SUCCESS. Sin embargo, debido a que el padre de P7 también es el padre de P6, P3 retornará 1 a P1, requiriendo revisar estas posiciones.

Nótese que P1 sólo debe revisar cada posición par con la siguiente; no necesita revisar la posición 1 contra la posición 2.

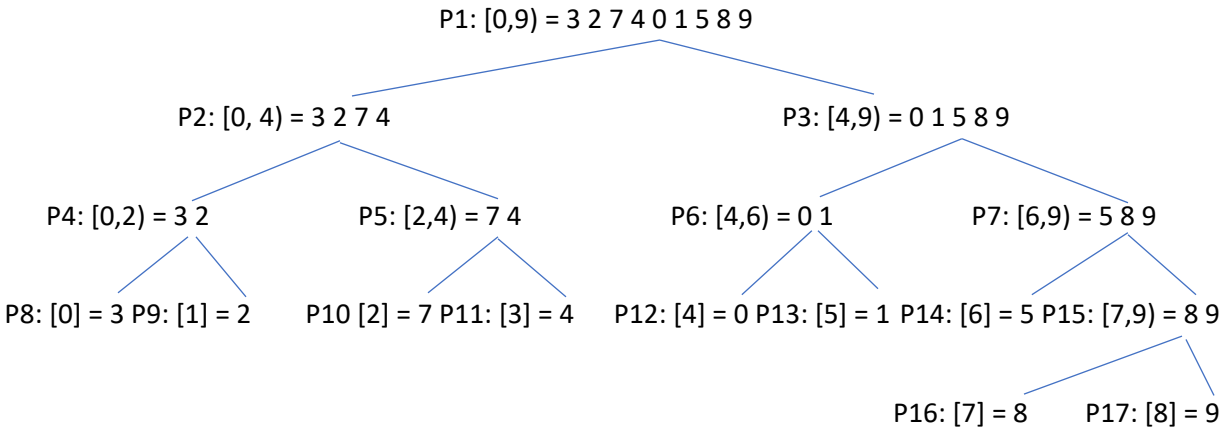


Luego de esta revisión, el arreglo queda 3 7 2 4 0 5 1 8 9, por lo que el segundo árbol de procesos sería:



En esta ocasión, todos los procesos de segundo nivel retornan EXIT_SUCCESS, confirmando que se han ordenado correctamente. Esto deja a P2 y P3 la tarea de revisar las fronteras: A[1]=7 contra la A[2]=2 (donde A es el arreglo), y A[6]=5 contra A[7]=1, respectivamente. Ambas revisiones fallan: no están en

orden, por lo que P2 y P3 retornan 2, indicándole a P1 que debe intercambiar la posición inicio + 2 con inicio + 1. Así, el tercer árbol de procesos sería:



En este caso, P4 y P5 fallan, retornando 1, requiriendo que se vuelvan a revisar las posiciones 0 vs 1, y 2 vs 3 del arreglo.

El programa sigue iterando con las siguientes listas:

- 2 3 4 7 0 1 5 8 9
- 2 3 4 0 7 1 5 8 9
- 2 3 0 4 1 7 5 8 9
- 2 0 3 4 1 5 7 8 9
- 0 2 3 4 1 5 7 8 9
- 0 2 3 1 4 5 7 8 9
- 0 2 1 3 4 5 7 8 9
- 0 1 2 3 4 5 7 8 9

4 Requisitos del programa

Cree un programa que reciba un arreglo de números y lo ordene usando el algoritmo descrito. El programa debe generar al menos una impresión por cada proceso creado. Para separar las impresiones, agregue un `sleep(…)` o `nanosleep(…)` cada vez que el proceso padre reordena el arreglo. Adicionalmente:

- Debe usar `fork(…)` y `wait(…)`
- Debe usar alguna función de la familia `exec(…)`
- Debe tener un `makefile`

5 Extra Credit

Puede presentar el siguiente programa al final del curso si le faltan puntos:

- Investigue los comandos equivalentes del API de Windows e implemente una aplicación equivalente que pueda correr en su *Command Prompt* (Símbolo de Sistema)
- Para puntos adicionales, implemente la aplicación de tal manera que pueda ser compilada en Windows o en Linux usando compilación condicional (`#if defined(WINDOWS)`, etc)